Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 1950 – 1958

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

Runtime Tracing of the Community Earth System Model: Feasibility Study and Benefits

Jens Domke[a], Dali Wang[b]

[a]*Joint Institute for Computational Sciences, Oak Ridge National Laboratory*
[b]*Environmental Sciences Division, Oak Ridge National Laboratory*

Abstract

The Community Earth System Model (CESM) is one of US's leading earth system modeling frameworks, which has decades of development history and was embraced by a large, active user community. In this paper, we first review the software development history of CESM and we explain the general objectives of performance analysis. Then we present an offline global community land model simulation within the CESM framework to demonstrate the procedure of runtime tracing of CESM using the Vampir toolset. Finally, we explain the benefits of runtime tracing to the general earth system modeling community. We hope those considerations can be beneficial to many other modeling research programs involving legacy high-performance computing applications.

Keywords: cesm, earth system model, tau, tracing, vampir, vampirtrace

1. Introduction

The Community Earth System Model (CESM) administrated by the National Center for Atmospheric Research (NCAR) is one leading US earth system model. The original version of CESM was created in the 1980s as the Community Climate Model (CCM). During the next two decades it was steadily improved and was renamed to Community Climate System Model (CCSM) after the Climate System Model components were introduced in the mid 1990s [1]. In 2004, NCAR released the third version of CCSM, which contained new versions of all component models. In 2007, this version of CCSM (called CCSM3) was used in the Intergovernmental Panel on Climate Change (IPCC) Fourth Assessment Report, alongside many other global climate models from different countries and institutions. In 2010, a new version of CCSM was released, and further evolved into the new Community Earth System Model (CESM). Currently, CESM contains five major community model components, atmosphere, ocean, sea ice, land, and land ice sheet, as well as associated data models to simulate earth systems.

Performance analysis of the CESM has been conducted along with the development of each individual physical (atmosphere, land, ocean, etc.) and software component (coupler, parallel I/O). For example, the source code of CESM (version 1.0), contains a general purpose timing library (GPTL) [2] and interfaces to the Performance API library (PAPI) [3]. Herein, we present our own method, which explores the feasibility and benefits of using a runtime tracing library to accomplish the performance analysis.

Email addresses: domkej@ornl.gov (Jens Domke), wangd@ornl.gov (Dali Wang)

doi:10.1016/j.procs.2012.04.213

In the following sections, we first present an offline global community land model simulation within the CESM framework, then we explain the detailed procedure of runtime tracing of CESM using the Vampir toolset. After that, we design several computational experiments to explore the performance of the offline global community model simulation on a Cray XK6. Finally, we explain the opportunities and impacts of runtime tracing to the general earth system modeling community.

2. Offline Community Land Model Simulation within the CESM Framework

The whole system of CESM is reconfigurable, which provides a flexibility to the community to design their own computational experiments. Herein, we reconfigure CESM into an offline global community land model simulation. Scientifically, this configuration uses historical climate forcings to drive the active land component simulation, and provides unique capabilities to verify and calibrate the land modeling activities with observational datasets.

The model components of the offline global community land model simulation contain a data atmosphere model, the active Community Land Model, and stub ocean, ice, and glacier models. The data atmosphere model reads in climate forcings to drive the active Community Land Model, version 4 (CLM4) [4]. CLM4 includes a complete prognostic treatment of surface energy, water, carbon, and nitrogen fluxes and state variables for both vegetated and non-vegetated land surface. Recent model development includes improved surface energy partitioning and thus water cycling [5, 6], and an improved ability to reproduce contemporary global patterns of burned areas and fire emissions [7]. Sub-surface hydrology parameterizations have also been changed to improve prediction of permafrost dynamics [8]. The coupled biogeochemistry functionality of the carbon and nitrogen cycle of CLM4 is an optional component, designated CLM-CN. All land modeling efforts described in this paper are performed with activated CLM-CN. The data exchange (including mass, energy, etc.) between each of those system components is enabled by a coupler [9], which is a software component that simplifies and expedites system integration. CESM utilizes the Network Common Data Form (NetCDF) for model input and output. Other CESM software components include an application driver, timing utility, parallel I/O [10], etc. CESM also provides a complex structure of shell scripts, which allow science-oriented modelers to reconfigure, compile, build, and submit jobs in an automatic fashion. It is an extremely valuable feature for CESM users to conduct computational experiments on a variety of high-performance computers [11]. We will show how to enable event tracing for the CESM framework in the following section.

3. Runtime Tracing of the CESM

3.1. Gathering of Performance Data

One step in performance analysis is the gathering of performance data itself. Depending on the level of detail you need or want to see, you may use different technics. Sampling or profiling have usually a low overhead, but have also a lack of detail. Sampling, e.g. with CrayPat [12] or the HPCToolkit [13], allows a general overview on how many time is spend in different functions and could reveal an estimated performance of the program sections while reading performance counters at the sampling points.

A more accurate method is profiling, where statistical information are gathered, like number of function calls and mean runtime of specific functions. GProf is one example for profiling the function calls of the program. Another example is the Integrated Performance Monitoring (IPM) tool [14], which is designed to profile MPI, OpenMP and file I/O calls. This can be combined with the analysis of PAPI counter for the program. As discussed in Section 1, the CESM has integrated profiling methods. This has three disadvantages, the first is the programming overhead while developing new routines or maintaining old code sections. Second, as long as not all functions are instrumented, the information are limited. The last disadvantage is, that profiling cannot be applied on subsections of the already instrumented source code without mayor changes in the build system or program code.

Even more information, compared to sampling and profiling, can be collected with event tracing. Instead of recording statistical information about events in the program code, the events and associated attributes are saved. An example is a MPI send operation with its begin and end timestamp and attributes, like communication partner, buffer size, etc. Hence, the exact program behavior can be reconstructed and analyzed. Toolsets like Scalasca [15] and its EPILOG trace format or TAU [16] are designed to monitor and analyze large-scale programs on a high level of detail. A further representative for tracing tools is the Vampir framework, which uses the Open Trace Format (OTF) [17] to

save the event data. We will introduce the Vampir framework in the next sections and show how we used the framework on CESM to gather information for further performance tuning.

3.2. VampirTrace

One component of the Vampir framework [18, 19] is VampirTrace (VT). The purpose of VampirTrace and its tools is to instrument the source code of the program and to manage the recording of events while the instrumented program is running. VampirTrace supports several programming languages. In detail they are C and Fortran, which is important for the tracing of CESM, and C++ and Java. The instrumentation can be done manually by the programmer, via compiler flags or with dynamic instrumentation of the binary. A detailed explanation on how these methods work can be looked up in the Vampir overview paper written by A. Knüpfer, et al., 2008. The fourth method of instrumentation is the usage of the TAU instrumentor to insert the necessary function calls into the source code. We used the latter method to be more flexible in terms of which functions or source files we want to instrument and to overcome the number of functions in the CESM source code, which cannot be instrumented manually within a reasonable time. We will specify this in Section 3.4. During the run of the instrumented program the recording library is responsible for processing the events and for transmitting them to the OTF library. The OTF library saves the events as an OTF record in an event stream. One event stream is written for each MPI process, OpenMP thread, POSIX thread and/or other thread, which is executed on an accelerator, like GPUs or Cell's SPEs [20].

3.3. Vampir

The Vampir framework includes a second component, which can be used to visualize the data/events gathered during the program execution. This component can be subdivided into VampirClient and VampirServer. The server analyzes the event streams of one or more traces simultaneously. The client puts the information of the server into graphs. Therefore the client provides a variety of displays, like the "Master Timeline" or the "Communication Matrix View", and functionality, like zooming and filtering. We will show some of the displays in Section 4 in context with the tracing of CESM. The general benefit of the client/server model is the capability to process trace files in parallel, i.e. multiple server processes can be used for one or more traces. This parallel processing overcomes the challenges [18], which are the product of the growing number of nodes, CPUs and cores of the current HPC systems.

3.4. The configuration of CESM to use VampirTrace

The user is faced with three major challenges while tracing an application. The first one is to figure out what type of information should be collected, i.e. which information is needed for further understanding, performances analysis and/or rewriting of parts of the application. The next challenge is the tradeoff between the amount of information needed and the amount of data written to the trace files. In general, it is easy to gather every information one can get from an instrumented application, i.e. by using the automatic compiler instrumentation, mentioned in Section 3.2. But in doing so, the trace streams might grow to a non-processable size, i.e. terabytes of data per stream. The third challenge is the runtime behavior of the application. Each traced event adds a small timeframe to the runtime [18]. A far too detailed instrumentation on the one hand could lead to a large overhead of more than 100% for the runtime and on the other hand could lead to a different program behavior, e.g. bottlenecks or imbalances become invisible. In the following we will address these challenge for CESM and show how we solved the problems.

In the first place we are interested in the MPI pattern of CESM to look for computational imbalances. Additionally we wanted to analyze the I/O behavior and the performance of the computation. To enable this, we have to change the compiler for the CESM framework in the Macros.<casename> file:

```
FC := vtf90 -vt:f90 ftn -vt:mpi -vt:inst manual
CC := vtcc -vt:cc cc -vt:mpi -vt:inst manual
```

The default Cray compiler wrappers on the Cray XK6 are exchanged with the VT wrapper. Since the VT wrappers do not recognize the Cray wrappers as MPI compilers without the '-vt:mpi' flag, it is necessary to enable MPI tracing via '-vt:mpi'. The '-vt:inst manual' flag tells the VT wrapper that the source code contains manual instrumentation, i.e. calls to the VT library, and that it is not necessary to invoke the automatic compiler instrumentation. Despite the use of manual instrumentation, MPI calls are traced. Therefore, even without any manual instrumentation of the source code, we see the MPI pattern of CESM. The additional tracing of I/O calls and PAPI counter (floating point operations per

second, total cache misses for L2 cache, data cache accesses for L2 cache) is enabled via VampirTrace environment variables in the env_mach_specific file of CESM:

```
setenv VT_IOTRACE 'yes'
setenv VT_METRICS 'PAPI_FP_OPS:PAPI_L2_TCM:PAPI_L2_DCA'
setenv VT_BUFFER_SIZE 512M
```

Supplementary, we increase the default trace buffer size (which is 32 MiByte) to 512 MiByte, so that no intermediate buffer flush will disrupt and influence the application execution. The results of this trace configuration can be seen in Section 4.1.1, 4.1.3 and 4.1.4 for the test case on 48 cores and in Section 4.2 for the one-year run on 240 cores.

Our next interest is the call graph of CESM and the differentiation of the components, e.g. land, atmosphere, etc. Using compiler instrumentation for the complete CESM source code does not work. The reason is a large number of very small subroutines, which are called millions of times. They produce too much trace data and the overhead is too high to make a meaningful statement on the performance of CESM. The filtering capabilities of VampirTrace might reduce the data, but the calls to the VT library are still present in the source code. Therefore, we cannot reduce the runtime overhead as needed, refer to Section 4.8 of [18] for details. For this reason, we use the TAU instrumentation, where we are able to prevent functions and complete source files from being instrumented. In the first iteration we used the TAU instrumentation for the complete source code, and the profiling mode of VT, to get the number of function calls per function. The profiling mode was enabled via the environment variable VT_MODE=STAT. For the next iteration and final trace we generated the tau.selective file to filter functions and complete source files, like the timing routines of the GPTL library. We used the profiling mode to identify and to filter all function with more than 5,000 calls per process. To use the tau.selective file together with the TAU instrumentation of VT, we have to change the compiler to the following:

```
FC := vtf90 -vt:f90 ftn -vt:mpi -vt:inst tauinst -vt:tau -f -vt:tau tau.selective
      -vt:cpp fpp -vt:preprocess
CC := vtcc -vt:cc cc -vt:mpi -vt:inst tauinst -vt:tau -f -vt:tau tau.selective
```

One problem was the TAU instrumentor, which produced incorrect instrumentation for Fortran code, when user-defined macros are preset directly after the variable declaration. We solve this by using a shell script, called fpp, which calls the preprocessor of our compiler and redirects the output to a specific file, defined by VT. Another problem were the build scripts of CESM, which removed the apostrophes from the original VT flag: -vt:tau '-f tau.selective'. A patch of the VT wrappers enabled the possibility to split this flag into two flags. The wrapper composes these two flags internally and passes them to the TAU instrumentor. In addition, we generated a VT group specification file, which was loaded via the environment variables like before: setenv VT_GROUPS_SPEC vt.groups. This group specification enables the possibility to gather all function of one component in a group with its own color, to distinguish the different components in the trace. We will show the result of this approach in Section 4.1.2.

4. Computational Experiments and Results

In this section, we present two computational experiments. The first one is a short term (two-day) simulation with 48 cores, which provides the opportunity to investigate the computational intensity, message passing pattern, call graph, as well as I/O pattern. The second case is a one-year simulation using 240 cores to investigate the communication and computational changes associated with ecosystem responses along with seasonal changes.

4.1. Short-term (2 days) simulation using 48 cores

CESM contains a straightforward way to reconfigure the parallel environment, i.e. number of MPI processes and number of OpenMP threads, for each component. The first case is designed to compare the efficiency of MPI-only code and MPI/OpenMP-hybrid code. The result, shown in Fig. 1, reveals that OpenMP is not fully enabled in the data atmosphere model. Thereby the overall execution time for the MPI-only case is shorter. The reason is, that one MPI process has to initialize the same amount of data for the hybrid case, which is initialized in parallel by four MPI processes for the other case. For the land model, the performance of both configurations looks different, but does not vary dramatically in terms of the runtime. Since the data atmosphere model is not OpenMP parallelized, we use all 48 cores for MPI processes in all following cases. The stubs for ocean, ice and glacier are executed on one core.

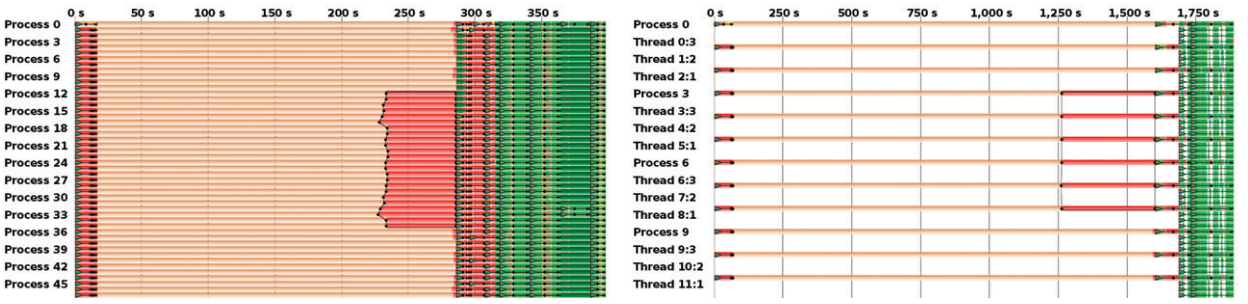

Figure 1: Left: Master timeline of the MPI-only case (duration: 398 s); Right: Master timeline of MPI/OpenMP-hybrid simulation (duration: 1886 s; data atmosphere model is not OpenMP parallelized); Coloring: illustrated in Fig. 4

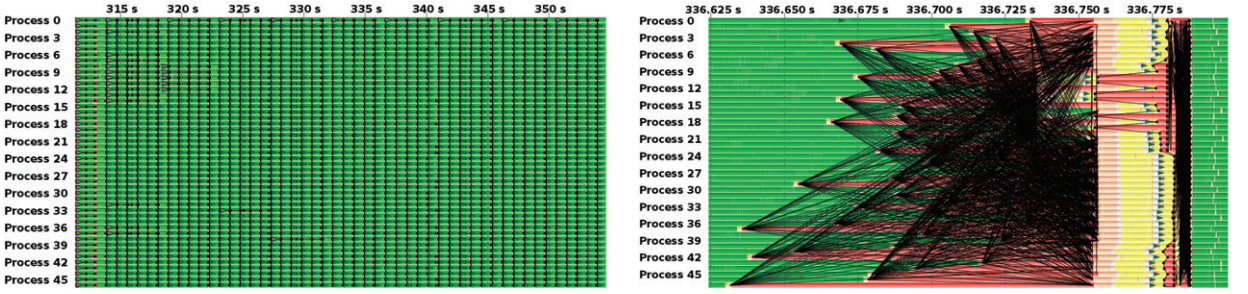

Figure 2: Left: MPI communication pattern of the land simulation for 24 h; Right: communication pattern of the coupler after one time step (zoom-in to one black vertical line of the left graph)

4.1.1. Communication pattern of the MPI simulation

The left graph of Fig. 1 is the overview of the whole simulation, and the left graph of Fig. 2 shows a zoom-in to the land model simulation. From the left graph of Fig. 1, two characteristics are obvious. First, the simulation starts with the climate data processing at process 0, which indicates that the parallel I/O is not enabled in the data atmosphere model. Second, the long waiting time of MPI_Allreduce for the processes 12–35 is caused by the static computational domain partitioning, which is based on the land mask. From the left graph of Fig. 2, it is clear that the MPI communication only happens at the end of each time step (30 min time frame) via the coupler. Fig. 2, right, shows the MPI communication pattern between two of these 30 min time frames. It is designed to enable the potential flux exchange between land and atmosphere. Since there is no communication required for the land model driver and biogeophysical and biogeochemical routines, the complete communication pattern is defined by the flux coupler. This involves extensive global communications, shown in Fig. 3.

4.1.2. Call graph and function grouping

We grouped all function calls according to the source module. Fig. 4 shows the accumulated exclusive time per function group, such as shared components, MPI, physical component (atmosphere and land), and auxiliary utilities. It is obvious that the MPI communication time is significant. Fig. 5 shows the call graph with the inclusive time for the function calls on root process 0. The common depiction of a call graph uses exclusive times, i.e. a bar will be drawn when the program executes instructions of the corresponding function. We use the call graph with the inclusive times, because we gain a general overview of the simulation, especially of the distribution into different components. From this graph, we can see that the data atmosphere model spends most of the time on interpolating climate forcings (solar radiation, precipitation, and surface temperature, pressure, etc.). It is evident that the major software complexity, in terms of call stack depth, comes from the simulation time manager inside of the coupler.

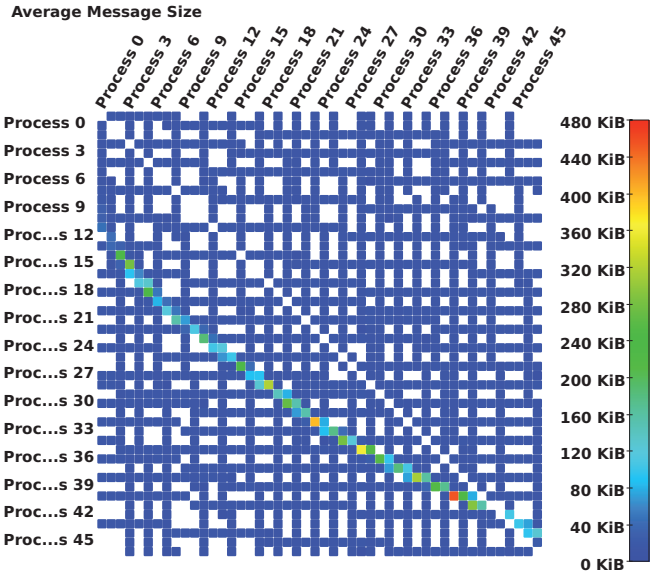

Figure 3: Comm. matrix of flux coupler (from Fig. 2, right)

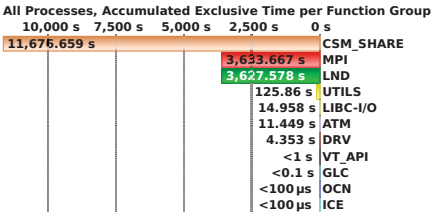

Figure 4: Grouping according to source module

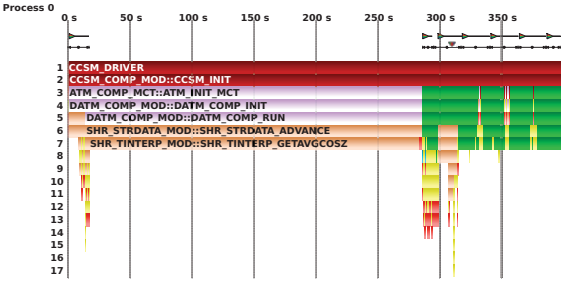

Figure 5: Call graph of process 0 (inclusive time)

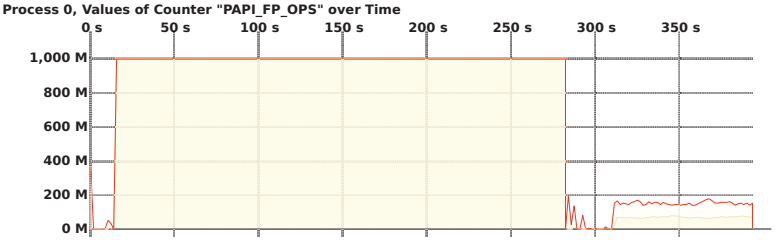

Figure 6: PAPI counter (flop/s) on root process 0

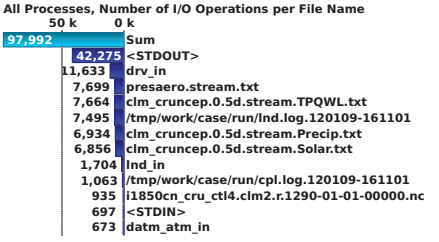

Figure 7: Summary of I/O statistics

4.1.3. Hardware Performance Counter

As shown in Fig. 6, the flop/s performance changes dramatically during the simulation. The simulation starts with I/O preparation and operations. After that, we see the peak performance for the data atmosphere model, which is ≈ 1,000 Mflop/s. The peak performance of the land model simulation is around 180 Mflop/s. It is also interesting to see that the average flop/s performance during the land model simulation is around 100 Mflop/s (see Fig. 8), which is mainly determined by the computation of the ecosystem behavior. Further performance characteristics of the land model can be looked up in Section 4.2, where we investigate the inter-annual changes and changes during the day.

4.1.4. I/O Activities

Fig. 7 shows the I/O statistics of the short-term simulation. From the graph it follows that the simulation uses ASCII data files to locate the input dataset. In addition, there is a detailed logfile for each component, including atmosphere, land, and coupler. Other than that mentioned, drv_in contains the general information about the simulation, i.e. case name, flux exchange rate, parallel environmental configuration, etc. The lnd_in file contains the input filename for the land component. The restart file for our simulation is i1850cn_cru_ctl4.clm2.r.1290-CN-01-0000.nc. The most I/O operations are done via STD_OUT, which is used to record the runtime status of the whole simulation. Although I/O has been considered as one potential bottleneck of the parallel CESM simulation, our study shows that I/O is not the bottleneck of global offline CLM simulation, see accumulated exclusive time for LIBC-I/O in Fig. 4.

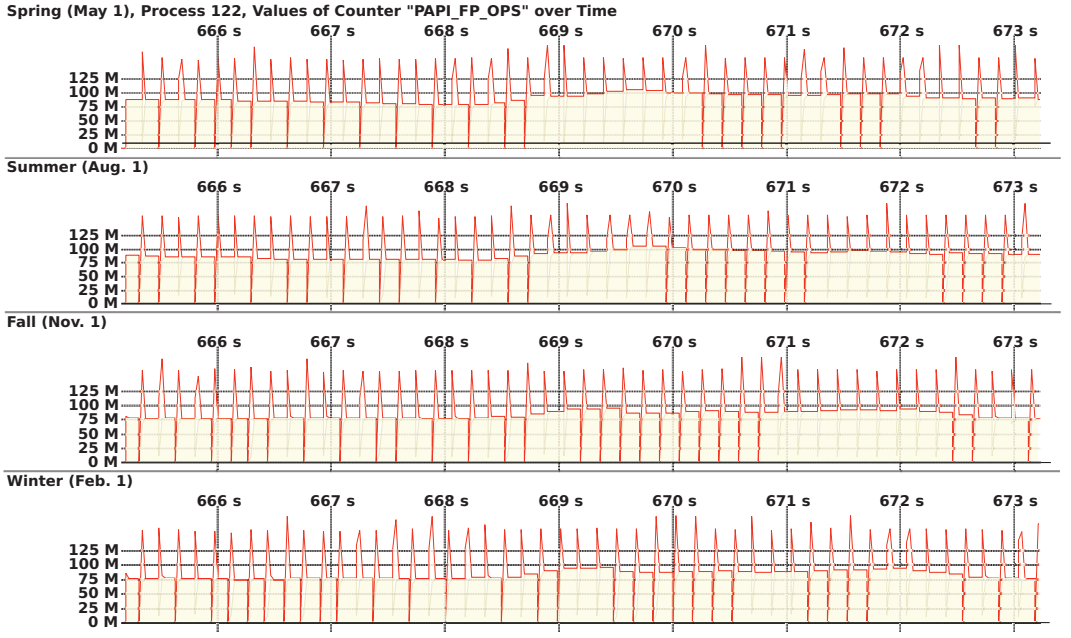

Figure 8: Flop/s for one day in different seasons. The performance spikes in those figures belong to the coupler.

4.2. Long-term simulation with a 240 core configuration

In this section, we design a one-year simulation to investigate the overall computation and communication ratio and the computational intensity changes along with inter-annual changes. Our method can be described as follows: we will make a two-month cold start run from January 1, 1850 to generate an initial restart file for the first trace run. The restart file of the two-month run will be taken for the spring simulation (Mar. 1 – May 31). After the spring season a new restart file for the next season will be generated. We generate traces for the remaining seasons accordingly: summer (Jun. 1 – Aug. 31), fall (Sep. 1 – Nov. 30) and winter (Dec. 1, 1850 – Feb. 28, 1851). The experiments are designed with the considerations of job queue policies at Oak Ridge National Laboratory, the expected trace size and our preset 512 MiByte buffer size limit for VampirTrace.

4.2.1. Communication vs. computation ratio

First of all, we look at the communication vs. computation ratio for the spring simulation. One fact is, that the ratio of $1 : 2.73$, i.e. 26.8% of the runtime was spent on communication, is not good for the complete simulation, and might reduce the speedup while increasing the number of cores. Most of the MPI time was spent in MPI barriers in the data atmosphere model and in context to I/O activities. Compared to this, the ratio is better, when the active land component calculates the climate changes for April 1850. In that case only 15.1% of the runtime was spent on communication. All the communications come from the explicit synchronizations required by the simulation time manager and shared software component. At the current stage of the geophysics and biogeochemistry implementation (without river routines), each land component cell could be computed independently, which would reduce the communication overhead.

4.2.2. Computational intensity

The global offline CLM simulation adapts a static domain decomposition. Therefore, the computational domain on each process is directly associated with a specific geographic location. Since the land system characteristics, such as photosynthesis, have a strong relationship with the climate forcings (such as solar radiation, temperature and precipitation, etc.), it is possible to identify the seasonal behavior changes via the computational intensity. Fig. 8 shows the flop/s on process 122, which is associated with a deciduous forest, for a 24 h time frame, from midnight to midnight.

As shown in Fig. 8, the computational intensity of ≈ 76 Mflop/s–96 Mflop/s in winter and fall is relative low compared to ≈ 80 Mflop/s–106 Mflop/s in spring and summer. In addition the computing intensity changes along the day, the lowest intensity is present in the wee hours and the highest intensity is present after noon. A similar trend can be found on several other processes. This feature has the potential to provide an opportunity to identify short-term climate extremes (like spring freeze), intensive precipitation or disturbances (fire) during the simulation. The reason is that those events will trigger different ecological processes, which have different computational intensity. Currently, it is common practice that monthly outputs (≈ 500 MByte each) are generated to provide information on inter-annual ecosystem variability. The monthly outputs may ignore the impact of short-term climate extremes. Therefore, the capability of detecting these extremes thru runtime tracing is valuable.

5. Summary and Conclusion

In this paper, we used VampirTrace to generate runtime event traces of the global offline Community Land Model simulation within the CESM framework. Afterwards, we used Vampir to visualize and analyze those trace files. We believe that runtime tracing is a helpful performance analysis method for the community earth system modeling, in addition to the current profiling utility of CESM. The benefits of runtime tracing, especially with the Vampir toolset in combination with TAU to instrument the functions, are the level of detail, which is adjustable without modifications of the source code, and the visualization and partial automatic analysis of the trace data. The activation of additional sources of information, like I/O tracing and hardware performance counters, during the tracing of the simulation enables the search for bottlenecks or examination of assumptions regarding performance and bottlenecks, see Section 4.1.1, 4.1.3 and 4.1.4.

From the performance toolset development perspective, the practice with the Community Earth System Model provides new opportunities to enhance the functionality and robustness of the Vampir toolset, especially related to long-term simulations of legacy HPC applications. We would not have been able to accomplish the analysis without the enhancements of VampirTrace during the preparation of the experiments, i.e. the support for I/O tracing on the Cray XK6 and the implementation of new flags for the VampirTrace wrapper to harmonize with the complex build system of CESM.

From the software engineering perspective, we see the runtime overhead produced by the implementation of the flux coupler, as well as the overhead of the timing management utilities. Considering that the computational intensity shifts along with season changes, mainly due to the ecosystem's responses to the climate forcings on spatial and temporal dimensions, we may need to adapt the dynamic load balancing during the simulation [21]. Furthermore, an OpenMP-parallelized implementation of the data atmosphere model would decrease the runtime of CESM simulations, which are not fully coupled.

Lastly, we would like to mention that the collaboration between performance tool developers and application developers is the key for this type of research on high-performance computers. We view this research as the start of a long-term collaboration. Besides the above mentioned dynamic load balancing, future work will include the performance analysis on other active components (atmosphere, ocean, ice and glacier), and a fully coupled CESM simulation.

Acknowledgments

The authors thank the Vampir Team of the Center for Information Services and High Performance Computing (ZIH), Technische Universität Dresden, for expertise and support during the time the tracing took place. This research was partially funded by Terrestrial Ecosystem Sciences (TES) Program and Climate Sciences for Sustainable Energy Future (CSSEF) Program under the Biological and Environmental Research (BER) under the Office of Science of the U.S. Department of Energy (DOE). This research used resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725. Oak Ridge National Laboratory is managed by UT-Battelle LLC for the Department of Energy under contract DE-AC05-00OR22725.

References

[1] J. B. Drake, I. T. Foster, Introduction to the Special Issue on Parallel Computing in Climate and Weather Modeling, Parallel Computing 21 (10) (1995) 1539–1544.

[2] J. Rosinski, General Purpose Timing Library (GPTL): A Tool for Characterizing Performance of Parallel and Serial Applications, presented at Cray User Group (May 2009).

[3] J. Dongarra, A. D. Malony, S. Moore, P. Mucci, S. Shende, Performance Instrumentation and Measurement for Terascale Systems, in: Proceedings of the 2003 International Conference on Computational Science, ICCS'03, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 53–62.

[4] K. Oleson, D. Lawrence, G. Bonan, M. Flanner, E. Kluzek, P. Lawrence, S. Levis, S. Swenson, P. Thornton, A. Dai, M. Decker, R. Dickinson, J. Feddema, C. Heald, F. Hoffman, J.-F. Lamarque, N. Mahowald, G.-Y. Niu, T. Qian, J. Randerson, S. Running, K. Sakaguchi, A. Slater, R. Stockli, A. Wang, Z.-L. Yang, X. Zeng, X. Zeng, Technical Description of version 4.0 of the Community Land Model (CLM), Tech. Rep. NCAR/TN-478+STR, National Center for Atmospheric Research, http://nldr.library.ucar.edu/repository/collections/TECH-NOTE-000-000-000-848 (Apr. 2010).

[5] D. M. Lawrence, P. E. Thornton, K. W. Oleson, G. B. Bonan, The Partitioning of Evapotranspiration into Transpiration, Soil Evaporation, and Canopy Evaporation in a GCM: Impacts on Land–Atmosphere Interaction, Journal of Hydrometeorology 8 (4) (2007) 862–880.

[6] R. Stoeckli, D. M. Lawrence, G. Y. Niu, K. W. Oleson, P. E. Thornton, Z. L. Yang, G. B. Bonan, A. S. Denning, S. W. Running, Use of FLUXNET in the community land model development, J Geophys Res-Biogeo 113 (G1).

[7] S. Kloster, N. M. Mahowald, J. T. Randerson, P. E. Thornton, F. M. Hoffman, S. Levis, P. J. Lawrence, J. J. Feddema, K. W. Oleson, D. M. Lawrence, Fire dynamics during the 20th century simulated by the Community Land Model, Biogeosciences 7 (6) (2010) 1877–1902.

[8] D. M. Lawrence, A. G. Slater, R. A. Tomas, M. M. Holland, C. Deser, Accelerated Arctic land warming and permafrost degradation during rapid sea ice loss, Geophysical Research Letters 35 (11) (2008) L11506+.

[9] J. Larson, R. Jacob, E. Ong, The Model Coupling Toolkit: A New Fortran90 Toolkit for Building Multiphysics Parallel Coupled Models, International Journal of High Performance Computing Applications 19 (3) (2005) 277–292.

[10] J. M. Dennis, J. Edwards, R. Loy, R. Jacob, A. A. Mirin, A. P. Craig, M. Vertenstein, An Application Level Parallel I/O Library for Earth System Models, International Journal of High Performance Computing Applications (IJHPCA), To Appear.

[11] D. Wang, W. Post, B. Wilson, Climate Change Modeling: Computational Opportunities and Challenges, Computing in Science Engineering 13 (5) (2011) 36–42.

[12] S. Kaufmann, B. Homer, CrayPat - Cray X1 Performance Analysis Tool, presented at Cray User Group (May 2003).

[13] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, N. R. Tallent, HPCTOOLKIT: Tools for performance analysis of optimized parallel programs, Concurrency and Computation: Practice and Experience 22 (6) (2010) 685–701.

[14] K. Fuerlinger, N. J. Wright, D. Skinner, Effective Performance Measurement at Petascale Using IPM, in: Proceedings of The Sixteenth IEEE International Conference on Parallel and Distributed Systems (ICPADS 2010), Shanghai, China, 2010, pp. 373–380.

[15] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, B. Mohr, The Scalasca performance toolset architecture, Concurrency and Computation: Practice and Experience 22 (6) (2010) 702–719.

[16] S. S. Shende, A. D. Malony, The Tau Parallel Performance System, The International Journal of High Performance Computing Applications 20 (2006) 287–331.

[17] A. Knüpfer, R. Brendel, H. Brunst, H. Mix, W. E. Nagel, Introducing the Open Trace Format (OTF), in: V. N. Alexandrov, G. D. van Albada, P. M. A. Sloot, J. Dongarra (Eds.), International Conference on Computational Science (2), Vol. 3992 of Lecture Notes in Computer Science, Springer, 2006, pp. 526–533.

[18] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, W. E. Nagel, The Vampir Performance Analysis Tool-Set, in: M. Resch, R. Keller, V. Himmler, B. Krammer, A. Schulz (Eds.), Tools for High Performance Computing, Springer Berlin Heidelberg, 2008, pp. 139–155.

[19] H. Brunst, A. Knüpfer, Vampir, in: D. Padua (Ed.), Encyclopedia of Parallel Computing, 1st Edition, Springer, 2011.

[20] D. Hackenberg, G. Juckeland, H. Brunst, High Resolution Program Flow Visualization of Hardware Accelerated Hybrid Multi-core Applications, in: CCGRID, IEEE, 2010, pp. 786–791.

[21] D. Wang, M. W. Berry, L. J. Gross, On Parallelization of a Spatially-Explicit Structured Ecological Model for Integrated Ecosystem Simulation, IJHPCA 20 (4) (2006) 571–581.